

Университет ИТМО

**Облачные системы и услуги**

**Лабораторная работа №**

Выполнили: Данилов Владислав Андреевич K34201**,**

Ледванов Вадим Дмитриевич K34201,

Смирнов Кирилл Андреевич K34201

**Преподаватель:** Мигулаева Татьяна Алексеевна

Санкт-Петербург

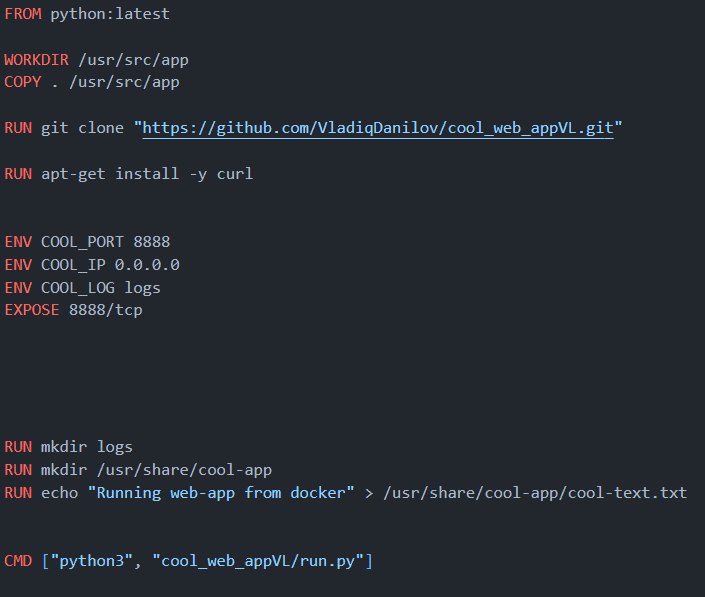
2024

**Цель работы:**

Изучить принципы написания Dockerfile и провести сравнительный анализ некорректных и оптимальных практик его разработки. Проанализировать две неэффективные практики, выделить их недостатки и на основе этого сформулировать выводы, способствующие повышению эффективности работы с контейнерами.

**Ход работы:**

**Задание 1**.

Был написан Dockerfile с Bad Practice. Его полный код ниже:

В нем создается основа образа для запуска Docker image: скачиваем Python с DocerHub, а далее - клонируем репозиторий с Web application, которое разворачивается следующим Python-кодом (в файле ‘run.py’):

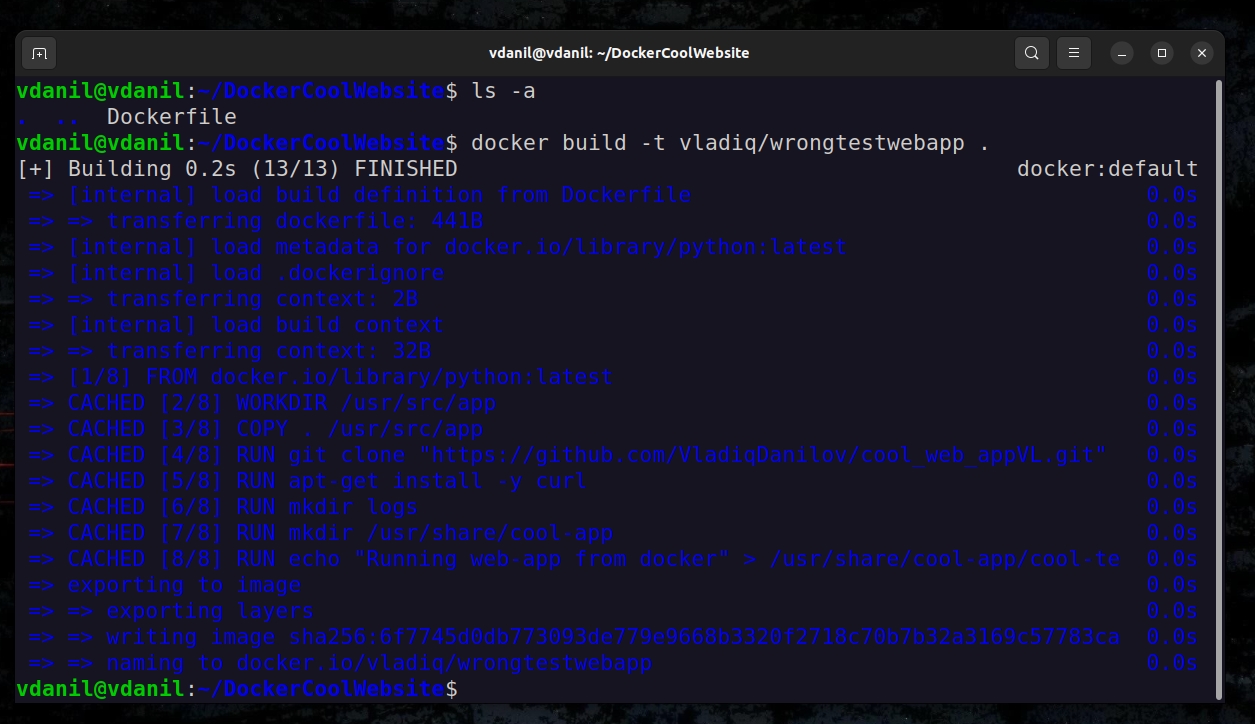


Устанавливаются библиотеки и обновляются зависимости.

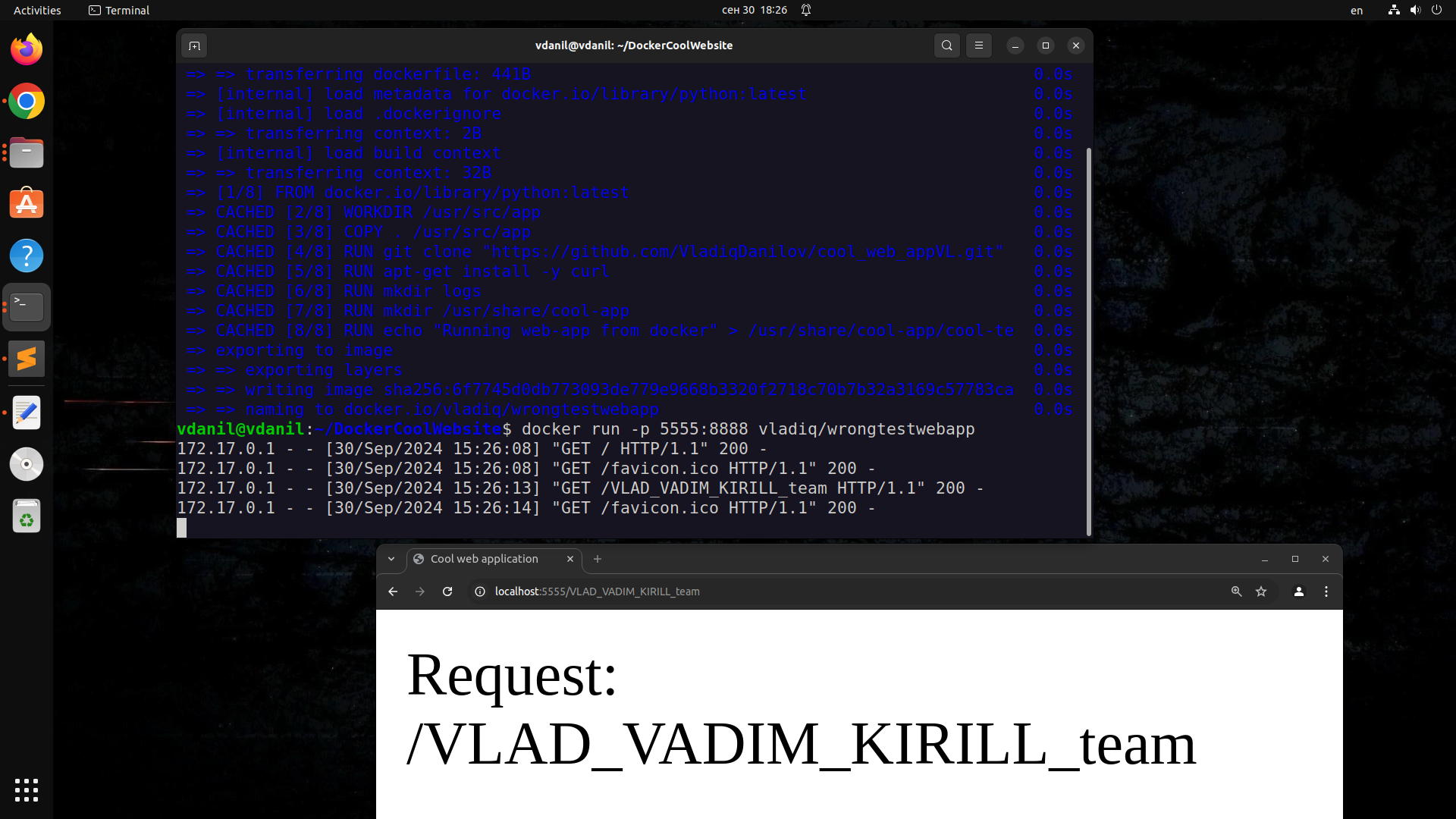
Создаются переменные окружения для внешнего localhost Docker контейнера: его IP и порт (COOL\_IP и COOL\_PORT соответственно) Далее, в переменной COOL\_LOG указывается директорию для хранения логов приложения, что позволяет контейнеру сохранять необходимые данные в указанное место для дальнейшего анализа.

Затем по адресу /usr/share/cool-app создаем файл cool-text.txt. С помощью оператора > записывается строка "Running web-app from docker" в файл, который выполняет функцию логирования начальной стадии запуска приложения. Это упрощает контроль за процессом и фиксирует успешный запуск контейнера.

Команда EXPOSE открывает указанный порт (в данном случае 8888) для взаимодействия контейнера с внешним localhost. В завершении, Уже на существующем контейнере запускаем cool\_web\_app/run.py при помощи CMD []

Запуск сборки контейнера с параметрами происходит следующим образом:

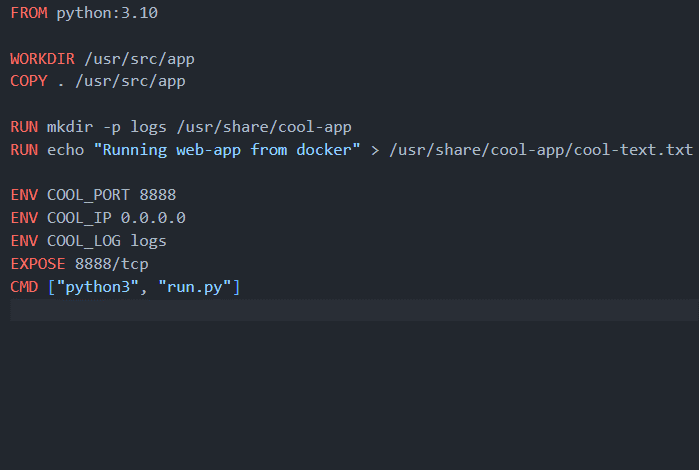
После окончания ‘Build’ можно проверить Собранное приложение в списке с помощью команды ‘docker ps’.

Производится запуск контейнера с приложением. Для взаимодействия с ним используется mapping портов (токен -p). Вид и функциональность приложения в докере представлены ниже:

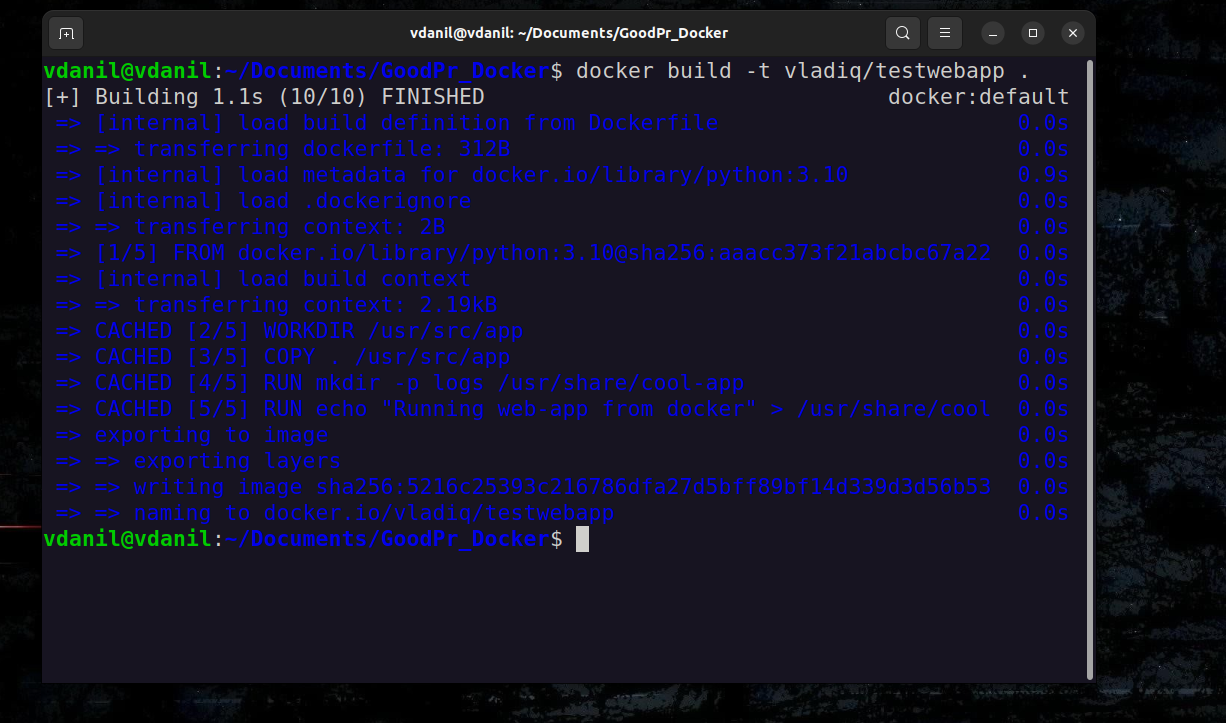
**Задание 2**.

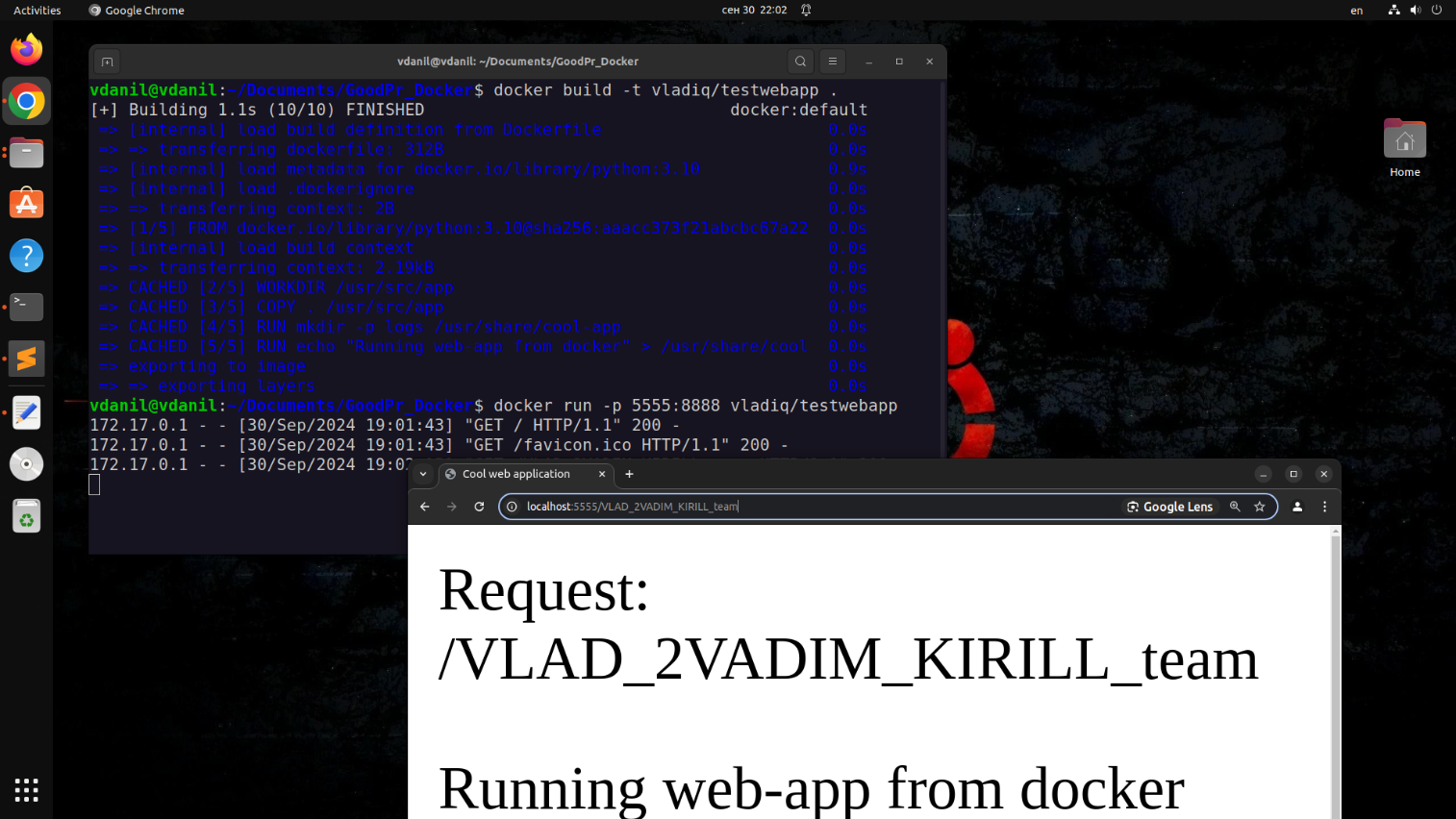
Хороший Dockerfile, был написан аналогично.

В нем вместо последней версии Python был явно указан образ Python:3.10 из Docker Hub. Вместо выполнения команды “RUN git clone” внутри Dockerfile, python-скрипт, необходимый для работы приложения, был скопирован напрямую из локальной директории хостовой машины с помощью команды “Copy”.

В данной версии Dockerfile вообще не требуется устанавливать пакеты, так как приложение не использует внешние зависимости. Проведена оптимизация создания директории. Полный код Dockerfile:

Запуск ‘Build’ Dockerfile:



Вид и функциональность приложения в Docker, созданный из обновленного Dockerfile представлены ниже:

**Задание 3**.

(Копия из Readme)

Плохие практики:

1) Использование последней версии Python для получения последнего Python Image из Docker Hub - это может привести к несовместимости в будущем

2) Использование внешней службы во время сборки (git clone в RUN команде).

С этим связано несколько проблем: git-репо может быть недоступен во время сборки, поскольку в этом случае происходит обращение к удаленному git хранилищу, т.е. никто не может гарантировать отсутствия проблем с сетью во время сборки Docker – образа.

Во-вторых, пользователь отдельной локальной Linux-машины может захотеть использовать этот же Docker-образ для другого git URL, а образ рассчитан под получение Python-скрипта через git. Следовательно, Docker не будет иметь точку входа.

3) Прямой запуск apt-get без update и clean в ‘RUN apt-get install -y curl’ может привести к большому информационному объему Docker-dir после Build

4) Создание директории и поддиректории для логов, хотя можно было бы сделать это в одном RUN

Как она (“bad practices”) была исправлена:

1) Docker image создается на основе python:3.10 из Docker Hub

2) Python-скрипт для создания будущего веб-приложения в Docker-среде копируется из текущей директории Dockerfile в контейнер

3) Не используется команда для update и install. В контесте данного Docker-приложения это и не нужно

4) Необходимые директории внутри контейнера создаются в одном месте ‘RUN mkdir -p logs /usr/share/cool-app’

**Задание 4**.

Описание двух плохих практик по работе с контейнерами:

1) Контроль ресурсов контейнера:

Если не следить за использованием процессора и оперативной памяти, то один контейнер может забрать слишком много ресурсов. Из-за этого другие контейнеры или OS на том же компьютере будут работать медленнее или перестанут работать совсем. Однако сами контейнеры были созданы, чтобы использовать ресурсы компьютера правильно. Так при правильном использовании и происходит.

Таким образом, во избежание этой проблемы нужно следить за ресурсами или установить ограничения на использование ресурсов при "Build" и "Run" контейнера.

2) Сохранение данных при работе с контейнерами:

Контейнеры представляют собой временные объекты, которые можно в любой момент удалить или перезапустить. Если данные приложения хранятся исключительно внутри контейнера, то при его перезапуске или удалении они могут быть утрачены. Это может привести к серьёзным проблемам. Поэтому нужны скрипты, логи и библиотеки должны хранится в локальной директории основной OS.

**Вывод:**

Был изучен процесс написания Dockerfile. Произведено сравнение “bad practices” и “good practices”.

На основе теории были определены решения, которые помогут разработчикам лучше создавать и управлять Docker-контейнерами.